# Integrating ACT-R and Cyc in a large-scale model of language comprehension for use in intelligent agents

**Jerry Ball, Stuart Rodgers, Kevin Gluck**

Air Force Research Laboratory, Human Effectiveness Directorate
Warfighter Training Research Division
6030 S. Kent Street, Mesa, AZ 85212-6061
FirstName.LastName@mesa.afmc.af.mil

**Abstract**

This paper describes a research program for the creation of a large-scale model of language comprehension for use in the development of language enabled intelligent agents. The language comprehension system is being implemented using the ACT-R cognitive architecture and modeling environment and adopts the cognitive principles and constraints of that architecture. The paper focuses on issues arising from the initial attempt to integrate the Cyc knowledge base into the language comprehension system as a means of scaling up the knowledge of the system to a size suitable for the development of intelligent agents.

## Introduction

First Newell (1980, 1990) and more recently Anderson and Lebiere (in press) have argued that both a vast knowledge base and natural language are important components of any unified, integrative account of human cognition. It remains a serious challenge in the cognitive modeling and agent representation communities to incorporate both of these into synthetic human representations that are psychologically principled and practically functional.

We are in the process of building a model of language comprehension that uses the ACT-R cognitive architecture and modeling environment coupled with the Cyc knowledge base to support the representation and processing of communications between synthetic agents and humans. The current version of the model (which does not yet integrate Cyc) is described in Ball (2003, 2004). Although the model is psycholinguistic in its theoretical orientation, it is intended to be a functional, large-scale model of language comprehension more in line with AI and Computational Linguistic approaches to Natural Language Understanding than typical computational psycholinguistic systems. To that end, we are investigating the possibility of integrating the large commonsense knowledge base provided in Cyc with the ACT-R implementation of the language processor. In this paper we focus on the mapping of Cyc terms and assertions to ACT-R declarative memory chunks and productions.

Within the computational cognitive modeling community, there have been a few attempts to integrate large knowledge bases into computational cognitive models of language processing. Bruno Emond (personal communication) integrated WordNet into an earlier version of ACT-R. The NL-Soar group (Deryle Lonsdale, personal communication) integrated WordNet into a NLP system implemented in Soar. Unfortunately, these efforts are not well documented and the WordNet integrations are no longer available.

Within the AI and Computational Linguistics community, there have been numerous integrations of large scale knowledge bases with language processing systems. The developers of Cyc have developed a commercially available language processing system which uses Cyc for its knowledge base. Many Computational Linguistic systems of language processing make use of large annotated corpora and knowledge bases (e.g. Nirenburg & Raskin, to appear; Wilks, Slator and Guthrie, 1996). Most of these integrations are engineering efforts and cognitive plausibility has not been a concern. As noted above, the focus of our research is on developing a large-scale language comprehension system, implemented in a cognitive architecture, and adhering to well known cognitive constraints on language comprehension. This difference in focus is a main reason we are not considering the wholesale adoption of any existing AI/Computational Linguistic systems, although our research is informed by such systems.

## Double R Model

The model of language comprehension discussed in this paper is called Double R Model (Referential and Relational Model). In addition to being implemented in the ACT-R cognitive architecture and modeling environment, Double R Model is founded on the linguistic principles of Cognitive Linguistics (Langacker, 1987, 1991; Talmy, 2001; Lakoff, 1987). Cognitive Linguistics is concerned with examining the relationship between form and meaning

American Association for Artificial Intelligence (www.aaai.org).

and argues against the existence of an autonomous syntactic component unlike Generative Syntax (Chomsky, 1965), which adopts the autonomy of syntax hypothesis and where form is studied in isolation from meaning. Double R Grammar is the Cognitive Linguistic theory underlying Double R Model. In Double R Grammar, the focus is on the representation and integration of referential and relational meaning—two key dimensions of meaning that get grammatically encoded. Double R Process is the psycholinguistic theory of language processing underlying Double R Model. Double R Process is a highly interactive theory of language processing which eschews a separate syntactic analysis feeding a semantic interpretation component in favor of a direct interpretation of the referential and relational meaning of input texts.

Double R Model is currently capable of processing an interesting range of grammatical constructions including: intransitive, transitive and ditransitive verbs; verbs taking clausal complements; predicate nominals, predicate adjectives and predicate prepositions; conjunctions of numerous grammatical types; modification by attributive adjectives, prepositional phrases and adverbs, etc. Double R Model accepts as input as little as a single word or as much as an entire chunk of discourse—using the perceptual component of ACT-R to read words from a text window. Unrecognized words are simply ignored. Unrecognized grammatical forms result in partially analyzed text, not failure. The output of the model is a collection of declarative memory chunks that represent the referential and relational meaning of the input text. The code for version 1 of the model is available on the Double R Theory web site at www.DoubleRTheory.com.

## ACT-R

ACT-R is a cognitive architecture and modeling environment for the development of computational cognitive models (Anderson & Lebiere 1998). It is a psychologically based cognitive architecture which has been used extensively in the modeling of higher-level cognitive processes, albeit on a smaller scale than that typical of AI/Computational Linguistic systems (see the ACT-R web site at http://act-r.psy.cmu.edu/ for an extensive list of models and publications). ACT-R is a hybrid system that includes symbolic production and declarative memory systems integrated with sub-symbolic production selection and spreading activation and decay processes. Production selection involves the parallel matching of the left-hand side of all productions against a collection of buffers (e.g. goal buffer, retrieval buffer, visual buffer) which contain the active contents of memory and perception. Production execution is a serial process—only one production is executed at a time. The parallel spreading activation process determines which declarative memory chunk is put in the retrieval buffer. ACT-R provides built-in support for single inheritance of declarative memory chunks and limited, variable-based pattern matching. Version 5 of ACT-R (Anderson et al., in press) adds a perceptual-motor component supporting the development of embodied cognitive models.

All knowledge in ACT-R is represented as either declarative memory (DM) chunks or productions, both of which are symbolic. DM chunks are implemented using a frame-based representation consisting of chunk types, slots and values. Chunk types are named and may be organized into a single inheritance hierarchy which may have multiple roots (hence, multiple trees). Slots are named, but not typed. Slots are not organized into an inheritance hierarchy. Slot values may be typed chunks or strings. If the value is a chunk, that chunk participates in the inheritance hierarchy. Chunk types are defined before model execution. Instances of chunks of a given type may be defined statically before model execution or created dynamically during execution. It is a general principle of the ACT-R architecture that chunks should be limited to 3 or 4 slots containing embedded chunks for their values, consistent with psychological constraints on the ability to simultaneously entertain multiple chunks, although this principle is not enforced by the architecture. Unlike productions, DM chunks are not supposed to be directional and the spreading activation mechanism of ACT-R can activate chunks in declarative memory via their slot values as well as using the chunk type to support retrieval.

In ACT-R, productions are procedural representations of skilled behavior that are not open to conscious reflection. They are symbolic, rule type structures with an antecedent and a consequent. The antecedent contains specifications that are matched against a collection of buffers containing DM chunks and perceptual information, and the consequent may alter the values of the slots in the DM chunks specified in the consequent or cause various actions to be performed. Typically, the antecedent of a production will match to the goal buffer, and optionally to the retrieval buffer, and/or one of the perceptual input buffers (e.g. visual buffer, auditory buffer). Variables are allowed in productions and can match to chunks, but variables cannot unify with other variables. Typically a variable in the antecedent matches to a chunk in one of the buffers, and a variable in the consequent matches to a chunk or the same variable in the antecedent. Variables are not allowed to be unbound across productions and must be instantiated within the production. Productions are inherently directional in ACT-R which uses a forward chaining inference method – i.e. the antecedent is always matched to the buffers during production selection and the consequent of the production is executed after selection.

The directional nature of productions in ACT-R is supported by empirical evidence about how humans perform procedural tasks that are highly automated (i.e. performing an automated procedural task backwards is much slower and more error prone than performing the task in the normal direction). Productions tend to be highly specific to the task being performed, but the existence of

generalized productions is not precluded. In the language comprehension system, generalized productions are used to effect default actions when no more specialized production matches the buffers. For example, there is a generalized production to retrieve the previously created DM chunk, if no other production matches the current buffer context.

## Cyc

Before selecting Cyc as our primary knowledge base, we reviewed several ontologies that are published online, including Cyc, WordNet, Generalized Upper Ontology, and others, to evaluate their relative strengths for this effort. Of the ontologies reviewed, Cyc and Wordnet had by far the largest number of elements defined or implemented. However, Cyc has the largest number ($10^6$) of axioms defined with the ontology's concepts. Primarily for this reason, we selected Cyc for our project.

Cyc is a very large general knowledge base of commonsense knowledge developed over much of the past 20 years by Doug Lenat and his team at Cycorp in Austin, Texas (Minsky, 2003; Lenat, 1995). The initial motivation for the project was to facilitate the development of reasoning systems which were knowledge intensive. Knowledge is embodied within the Cyc ontology.

The Cyc ontology has two fundamental expression types: terms and assertions. Terms can be either atomic (individual constants) or non-atomic (a function with arguments). Individual constants provide names for concepts and the relations between concepts. Functions allow for the automatic creation of many non-atomic terms such as *(LiquidFn Nitrogen).* Terms categorize the stuff and things of everyday life such as *dogs*, *weather*, *dates*, *ideas*, *people*, etc. Terms can also be logical connectives such as *and* or *implies*. Terms are combined to make various types of assertions. The primary relational term in an assertion is the predicate. Predicates relate things or concepts to other concepts for example: (*fatherOf Cain Adam)*. This relation asserts that Cain's father is Adam. Predicates relate individuals to groups, classes to instances, opposites, generalizations, and others, for example: *memberOf, isa, disjointWith,* and *genls.*

Cyc makes no distinction between procedural and declarative knowledge, representing both as terms and assertions. The Cyc inference engine provides inferencing capabilities based on logic, unification, forward chaining (the default) and backward chaining (when specially invoked), combined with additional capabilities based on a large number of specialized heuristics. Because of its vast size, Cyc has microtheories to support locally (i.e. within a microtheory) consistent assertions, but potentially globally inconsistent assertions.

## Core Similarities and Differences

Cyc was originally implemented using frames and subsequently converted to a logical representation when the frame based representation proved to be too awkward and inflexible (Whitten, 1995). In part, this stems from the inflexibility of embedding attributes within the slots of frames, where accessing the value of a slot requires access to the frame first. Logical representations do not have this limitation. The value of an attribute can be accessed directly, assuming the attribute is represented as a predicate and predicates are indexed for direct access. On the other hand, in a frame representation, once the frame is identified, the values of the slots can be retrieved without search. In a logical representation, the argument (i.e. the object) to which the predicate (i.e. attribute) applies must be indexed to avoid search. This can be expensive if multiple argument positions must be indexed. Note that implementations of Prolog (e.g. Quintus Prolog, Sicstus Prolog) often only index on the first argument of a predicate to avoid this expense, although Cyc indexes on all arguments. In ACT-R, although productions are explicitly directional, DM chunks are not meant to be. ACT-R provides sub-symbolic capabilities that mitigate the directional limitations of its frame based representations. In particular, the spreading activation mechanism of ACT-R allows the slot values of the DM chunk in the goal buffer to activate DM chunks in declarative memory with matching values, thereby providing an index-like capability to access DM chunks via their slots and not just via their chunk types.

Both Cyc and ACT-R support inheritance, however, Cyc provides a multiple inheritance capability, whereas ACT-R only provides built-in support for single inheritance. As discussed below, this has important ramifications for the mapping from Cyc to ACT-R.

From an object oriented programming perspective, neither ACT-R nor Cyc provide a mechanism for encapsulating methods within the DM chunks or terms that correspond most closely to objects. ACT-R provides an *eval* operator which makes it possible to insert lisp code into productions and Cyc provides an *evaluationDefn* predicate associated with functions that supports the inclusion of code to compute the function, but neither of these are encapsulated in the way typical of many object oriented systems. The slots in ACT-R's DM chunks correspond to the data values encapsulated in objects, but the slots are not typed, although the DM chunks which are the values of the slots are typed. In Cyc even attributes are not encapsulated. Rather, they are specified external to the concept they are attributes of, with a pointer to the concept included as an argument of the predicate that specifies the attribute.

Cyc was designed from the ground up with large-scale systems in mind. ACT-R was also designed to scale, but

has primarily been used in the development of small-scale cognitive models. The serial implementation of the parallel spreading activation and production selection processes may limit the ability of ACT-R to scale to the size of the Cyc knowledge base. This may not be an issue so long as the full Cyc KB is not loaded into ACT-R at runtime.

Cyc is a purely symbolic system. ACT-R provides sub-symbolic spreading activation, production selection, and stochastic noise mechanisms.

Cyc uses microtheories (for which ACT-R has no equivalent) to model large chunks of time, and has the dynamic event collections to support the modeling of dynamic events. ACT-R is inherently time dependent, but provides no means for reasoning about time. On the other hand, ACT-R has the capability to model the time course of cognition at the millisecond level. Cyc has built-in constraints on the duration of inferencing, but otherwise does not model the time course of cognition.

Whereas ACT-R uses specialized productions and forward inferencing which is very efficient, but not general, Cyc relies on a generalized logic based inferencing engine (forward by default, but backward, as well) that is made more efficient via the use of heuristic rules. It is with respect to their inferencing capabilities that Cyc and ACT-R differ most.

Despite the differences discussed above, ACT-R and Cyc have much in common as knowledge representation systems and their integration is feasible and desirable given the objectives of this research. Both are essentially propositionally based systems of symbolic representation with ACT-R opting for frame based propositional representations and Cyc opting for predicate/argument based propositional representations. Neither provides imagistic representation or non-propositional spatial processing capabilities.

## Mapping Cyc's Terms and Assertions to ACT-R's DM Chunks and Productions

Cyc terms may be straightforwardly mapped to ACT-R DM chunks. For example, the Cyc term *PhysicalDevice* (terms in Cyc are normally prefixed by *#$*, but the prefix is ignored below) can be mapped to the ACT-R DM chunk *physicaldevice*, where the DM chunk is defined by a chunk type definition that contains no slots.

The mapping of Cyc assertions to ACT-R is more complicated. Consider the mapping of Cyc's generalization (*genls*) and instance of (*isa*) assertions to ACT-R. One approach for mapping these assertions from Cyc to ACT-R would be to use ACT-R's built-in single inheritance mechanism to replicate the Cyc knowledge structure of generalization/specialization, and instances of concepts. For example, in Cyc, the concepts of *PhysicalDevice* and *Weapon* are related as a generalization/specialization (i.e. weapon is a class that is a specialization of the physical device class), and *Colt45Revolver* is represented as an instance of *Weapon* (i.e. colt 45 revolver is an instance of weapon). Note that the distinction between specialization and instantiation is not always clear cut, however, instantiation is intended to cover the relationship between a class and an instance of the class, whereas, specialization is intended to cover the relationship between a subclass and a class. In Cyc language, this is written as follows:

```
(genls Weapon PhysicalDevice)
(isa Colt45Revolver Weapon)
```

where *genls* stands for "generalization" and *isa* stands for "is an instance of" (with the more specific terms being listed first). The Cyc inferencing engine includes all the needed means for inheriting the attributes of parent concepts and for inferencing over these representations.

Using ACT-R's built-in single inheritance mechanism, these concepts could be defined as:

```
(chunk-type (weapon (:include physicaldevice) ... )
(colt45revolver isa weapon)
```

where *chunk-type* identifies a chunk type (the basis for defining DM chunks in ACT-R) *:include* indicates that *weapon* is a subtype of *physicaldevice*, … indicates that the chunk type may have additional slots defined as part of the chunk type frame. Once the chunk type is defined, instances of the chunk type (i.e. chunks) are defined using the *isa* chunk definition frame with *colt45revolver* naming the chunk and *weapon* identifying the chunk type. However, this mapping is problematic when a concept or object in Cyc inherits from more than one parent. For instance, in Cyc, *PhysicalDevice* is a specialization of both *Artifact* and *SolidTangibleProduct*. There is no built-in facility in ACT-R to inherit from two parent chunk types (despite the fact that the underlying Common Lisp language supports multiple inheritance). Therefore, the definition of *physicaldevice* in ACT-R would require a choice to inherit from either the *artifact* chunk-type or the *solidtangibleproduct* chunk-type, somehow accommodating the other parent.

Further, it is not clear that ACT-R's built-in single inheritance mechanism is appropriate in any case. In the tutorial that is provided with ACT-R, Unit 1 contains a model of semantic memory that does not use the built-in single inheritance mechanism. Instead inheritance is implemented explicitly in the model. From a programming perspective, this makes model creation more difficult, but the computational cognitive modeler has full control over the implementation of inheritance. Further, the use of the built-in single inheritance mechanism circumvents the ACT-R architecture's computation of cognitive processing time. That is, inferencing based on inheritance requires no cognitive processing time if the built-in inheritance mechanism is used.

Given this, we are working toward a more general solution to the problem which does not use ACT-R's built-

in single inheritance. Our method allows the relationship between *physicaldevice* and the more general concepts, *artifact* and *solidtangibleproduct*, to be defined as follows:

(chunk-type genls-ct child parent)
(achunk1 isa genls-ct child physicaldevice parent artifact)
(achunk2 isa genls-ct child physicaldevice parent solidtangibleprodujct)

In the *chunk-type* definition of *genls-ct* (i.e. generalization chunk type), the slots *child* and *parent* are defined. Then two chunks are defined to represent the child-parent relationships between *physicaldevice* and *artifact* and *physicaldevice* and *solidtangibleproduct*. This approach will cover the *genls* assertions in the Cyc knowledge base. A similar chunk type definition and chunks can be used to cover Cyc's *isa* assertions.

(chunk-type isa-ct instance class)
(achunk3 isa isa-ct instance colt45revolver class weapon)

The primary chunk types and chunks currently envisioned are the relationships *isa-ct, genls-ct [specialization]*, and *property-ct* plus the broad situation concepts *action-ct, event-ct, process-ct, situation-ct,* and *state-ct.* This approach with these primary chunk types will cover a large majority of the Cyc knowledge base.

In general, there is a very direct mapping from Cyc assertions to ACT-R DM chunks for those assertions that are simple predicate/argument structures where the arguments are simple terms. However, Cyc allows for more complex assertions including logical operators which result in more complex structures. In Cyc, these complex assertions are referred to as *rules* and the mapping to ACT-R is less straightforward.

Automating the mapping of Cyc rules into ACT-R introduces additional considerations. Should they be mapped to ACT-R DM chunks or productions? If they are mapped to DM chunks, then the assertions may need to be mapped to multiple DM chunks and productions will be needed to support inferencing over them. If they are mapped to productions, the generated productions will require appropriate handling of the logical operators in those rules. For example, conjunctions could be handled by including multiple chunks in the slots of the goal template that is associated with a production for each conjunct. Disjunction could be handled by using separate ACT-R productions for each disjunct. Logical implication can be handled by the basic antecedent/consequent nature of ACT-R's productions. ACT-R also provides a negation operator to negate a production rule slot value that matches to a buffer and this could be adapted to handle logical negation. However, ACT-R provides no quantification capabilities, so additional productions and DM chunks will be needed to handle quantification.

Besides the complexities in mapping Cyc rules to ACT-R, a method for handling the mapping of Cyc functions and microtheories to ACT-R is yet to be developed.

As the discussion above suggests, there are many subtle issues involved in the mapping of Cyc assertions to ACT-R DM chunks and productions, and fully automating this mapping will not be simple. However, if successful, we will be able to integrate the Cyc knowledge base with our ACT-R based language comprehension system, providing a huge increase in the knowledge available to support language comprehension.

## Some Additional Considerations

In their theory of sentence memory, Anderson, Budiu, and Reder (2001) adopt a representational system within the declarative memory component of ACT-R that contains only two types of chunks: nodes and links. Given only node and link chunks, the representation of a sentence like "Bob paid the waiter" necessarily consists of multiple chunks: node chunks to represent the individual words and phrases, and link chunks to represent the hierarchical structure of the sentence. The approach of Anderson et al. can be contrasted with a representational system in which a single chunk encodes the entire sentence (with the slots in the chunk containing the subcomponents of the sentence). Anderson et al. provide empirical support for their system of representation based on the fragmentary retrieval of parts of sentences in several memory retention studies.

From the perspective of frame-based representations, limiting the possible types of "frames" (or chunks) to just nodes and links may appear highly constraining. However, nodes and links are the basic components of network representations, and network representations are a highly general form of knowledge representation. In restricting the possible frame types to nodes and links, what Anderson et al. have effectively done is implement a network architecture within the frame based architecture of ACT-R's declarative memory component.

This network architecture resides within the over-arching ACT-R architecture and is influenced by that architecture in various ways. For example, ACT-R's spreading activation mechanism can activate nodes and links independently of the structure that the nodes and links combine to form via addition of an index to each node and link chunk. Anderson et al. call this index the "context" slot and use it to tie the components of a linguistic representation together. If all the components of a linguistic representation have the same value for the context slot, they will all be activated (to some degree) by a goal chunk containing the index as one of its slot values. Thus, the entire linguistic representation can be activated in parallel even though there are multiple chunks involved. Note that it is also possible to add an index to the components of a single chunk corresponding to the entire sentence, thereby activating the components of that representation without having to traverse the representation.

That is, combining an index with the spreading activation mechanism makes possible the activation of co-indexed chunks independently of any other structural organization. Thus, it is possible to combine a more structured frame-like representation with indexing/spreading activation as a way of providing alternative access to the components of a representation.

Switching to another consideration, although Cyc is a huge knowledge base of commonsense knowledge, it turns out that Cyc lacks the domain specific knowledge needed for the initial application of the language comprehension model: pilot communication. Pilot communication is highly specialized and the brevity terms that pilots use have specialized meanings that are not captured in the Cyc knowledge base. Given this, a separate knowledge engineering effort is required to add the domain specific knowledge needed before intelligent agents capable of acting as pilots can be developed. And this requirement brings to fore the knowledge acquisition capabilities of Cyc. It has been our experience that the knowledge input capabilities of Cyc are difficult to use without extensive knowledge and experience with the Cyc knowledge base. Deciding where in the Cyc ontology to attach domain specific knowledge is an important and difficult decision to make, and we do not fully understand the consequences of that decision. Should our domain specific knowledge be encoded as a separate micro-theory to avoid conflicts with the existing ontology, or do we want to integrate the domain specific knowledge into an existing micro-theory? More generally, what is the appropriate level of granularity for micro-theories? These considerations are made even more controversial since we have no theory of the mapping of Cyc micro-theories into ACT-R. Can we use an indexing mechanism like that employed by Anderson et al. (2001) to tie the assertions in a micro-theory together, or will that lead to an undesirable proliferation of indexes to tie together representations at different levels of granularity?

## Summary

We are attempting to integrate the Cyc knowledge base into ACT-R as a way of scaling up our language comprehension system for use in the development of language capable intelligent agents. With respect to the results of our effort, the jury is still out. We have defined the mappings needed to convert most of the terms and assertions in the Cyc knowledge base into ACT-R DM chunks and productions. However, Cyc does not provide the domain specific knowledge needed for our initial application and this domain specific knowledge will have to be manually added. Further, it is unclear how much of the knowledge that is provided in Cyc will prove necessary, given the specialized domain of our initial application. Despite these concerns, we are convinced that the development of functional language comprehension systems necessitates the integration of large sources of knowledge and Cyc is the largest such source currently available. In this respect we agree with Feigenbaum (2004) and his assertion that the acquisition and integration of large amounts of knowledge are the keys to the development of intelligent systems, and we are intrigued by his suggestion that AI researchers focus on the development of systems capable of acquiring knowledge by reading texts as a way of overcoming the knowledge acquisition bottleneck.

## References


Anderson, J. R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C., & Qin, Y. (in press). An integrated theory of the mind. *Psychological Review*. http://act-r.psy.cmu.edu/papers/403/IntegratedTheory.pdf

Anderson, J. R., Budiu, R., and Reder, L. (2001). A Theory of Sentence Memory as Part of a General Theory of Memory. *Journal of Memory and Language*, 45, 337-367.

Anderson, J. R,. & Lebiere, C. (1998). *The Atomic Components of Thought*. Mahway, MJ: LEA.

Anderson, J. R., & Lebiere, C. (in press). The Newell Test for a theory of mind. *Behavioral and Brain Sciences*.

Ball, J. (2003). Double R Theory. www.DoubleRTheory.com

Ball, J. (2004). A Cognitively Plausible Model of Language Comprehension. In *Proceedings of the Thirteenth Conference on Behavior Representation in Modeling and Simulation*. ISBN: 1-930638-35-3. Available on line at http://www.sisostds.org/conference/index.cfm?conf=04BRIMS (04-BRIMS-015).

Chomsky, N. (1965). Aspects of the Theory of Syntax. Cambridge, MA: The MIT Press.

Engelmore, R., and Morgan, A. eds. (1986). *Blackboard Systems.* Reading, Mass.: Addison-Wesley.

Feigenbaum, E. (2004). Keynote address. Florida AI Research Society Conference.

Lakoff, G. (1987). Women, Fire and Dangerous Things. Chicago: The University of Chicago Press

Langacker, R. (1987). Foundations of Cognitive Grammar, Volume 1, Theoretical Prerequisites. Stanford, CA: Stanford University Press

Langacker, R. (1991). Foundations of Cognitive Grammar, Volume 2, Descriptive Applications. Stanford, CA: Stanford University Press.

Lenat, D (1995). Cyc: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM* 38:11.

Minsky, M. (2003). "Why A.I. Is Brain-Dead." At *Wired Magazine* online, Issue 11.08 http://www.wired.com/wired/archive/11.08/view.html?pg=3

Newell, A. (1980). Physical symbol systems. *Cognitive Science, 4*, 135-183.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Nirenburg, S. & Raskin, V. (to appear). *Ontological Semantics*. Cambridge: The MIT Press.

Talmy, L. (2003). Toward a Cognitive Semantics, Vols I and II. Cambridge, MA: The MIT Press.

Whitten, D. (1995). Subject: [7] How does it work? http://www.robotwisdom.com/ai/cycfaq.html.

Wilks, Y., Slator, B., & Guthrie, L. (1996). *Electric Words: Dictionaries, Computers, and Meanings.* Cambridge, MA: The MIT Press.